Continuous Consensus with Failures and Recoveries

Tal Mizrahi and Yoram Moses

Department of Electrical Engineering, Technion, Haifa 32000 Israel
deweastern@yahoo.com, moses@ee.technion.ac.il

Abstract. A *continuous consensus* (CC) protocol maintains for each process i at each time k an up-to-date core $M_i[k]$ of information about the past, so that the cores at all processes are guaranteed to be identical. This is a generalization of simultaneous consensus that provides processes with the ability to perform simultaneously coordinated actions, and saves the need to compute multiple instances of simultaneous consensus at any given time. For an indefinite ongoing service of this type, it is somewhat unreasonable to assume a bound on the number of processes that ever fail. Moreover, over time, we can expect failed processes to be corrected. A failure assumption called (m,t) interval-bounded failures, closely related to the *window of vulnerability* model of Castro and Liskov, is considered for this type of service. The assumption is that in any given interval of m rounds, at most t processes can display faulty behavior.

This paper presents an efficient CC protocol for the (m,t) bound in the crash and sending omissions failure models. A matching lower bound proof shows that the protocol is optimal in all runs (and not just in the worst case): For each and every behavior of the adversary, and at each time instant m, the core that our protocol maintains at time m is a superset of the core maintained by any other correct CC protocol under the same adversary. The lower bound is a significant generalization of previous proofs for common knowledge, and it applies to continuous consensus in a wide class of benign failure models, including the general omissions model, for which no similar proof existed.

Keywords: Agreement problem, Consensus, Continuous Consensus, Distributed algorithm, Early decision, Common Knowledge, Lower bound, Modularity, Process crash failure, Omission failures, Process recovery, Round-based computation model, Simultaneity, Synchronous message-passing system.

1 Introduction

Fault-tolerant systems often require a means by which independent processes or processors can arrive at an exact mutual agreement of some kind. As a result, reaching consensus is one of the most fundamental problems in fault-tolerant distributed computing, dating back to the seminal work of Pease, Shostak, and Lamport [17]. In the first consensus algorithms, decisions were reached in the same round of communication by all correct processes. It was soon discovered, however, that allowing decisions to be made in different rounds ("eventual agreement") at different sites gives rise to simpler protocols in which the processes can often decide much faster than they would if we insist that decisions be simultaneous [5]. There are cases in which eventual agreement often suffices: In recording the outcomes of transactions, for example. In other

G. Taubenfeld (Ed.): DISC 2008, LNCS 5218, pp. 408–422, 2008.

cases, however, a simultaneous decision or action is often required or beneficial: E.g., when one distributed algorithm ends and another one begins, and the two may interfere with each other if executed concurrently. Similarly, many synchronous algorithms assume that the starting round is the same at all sites. Finally, there are cases in which the responses in a given round to external requests at different sites for, say resource allocation, must be consistent. A familiar application that assists in many of these is the Firing Squad problem [1, 4].

Motivated by [5, 9], the problem of reaching simultaneous consensus was shown in [7, 13] to require the correct processes to attain *common knowledge* about the existence of particular initial values. By computing common knowledge efficiently in the crash failure model, they designed protocols for simultaneous consensus that are optimal *in all runs*, and not just in the worst case: In every execution, they decided as fast as any correct protocol could, given the same behavior of the adversary.[1] Finally, they also observed that computing facts that are common knowledge, in essentially the same manner, can solve other simultaneous coordination problems such as the firing squad problem. While [7] considered crash failures, [13] extended the analysis of common knowledge to omission failure models. They designed an efficient protocol for computing all facts that are common knowledge at a given point, and used this to derive optimal protocols for a wide class of *simultaneous choice* problems. More recently, this work was further generalized in [11], where a general service called *Continuous Consensus* (CC) that serves to support simultaneous coordination was defined.[2] It is described as follows.

Suppose that we are interested in maintaining a simultaneously consistent view regarding a set of events $\mathcal{E}$ in the system. These are application-dependent, but will typically record inputs that processes receive at various times, values that certain variables have at a given time, and faulty behavior in the form of failed or inconsistent message deliveries. A *(uniform)*[3] *continuous consensus* protocol maintains at all times $k \geq 0$ a *core* $M_i[k]$ of events of $\mathcal{E}$ at every site i. In every run of this protocol the following properties are required to hold, for all processes i and j.

Accuracy: All events in $M_i[k]$ occurred in the run.
Consistency: $M_i[k] = M_j[k]$ at all times k.
Completeness: If e occurs at a process j at a point at which j is nonfaulty, then $e \in M_i[k]$ must hold at some time k.

The continuous consensus problem generalizes many problems having to do with simultaneous coordination. Using the core of a CC primitive, processes can independently choose compatible actions in the same round. Thus, the *firing squad* problem [4]

[1] We think of the adversary as determining the pattern of initial votes and the pattern in which failures occur, in each given execution of the protocol. The performance of a protocol P can be compared to that of P' by looking at respective runs in which the adversary behaves in the same way.

[2] A different problem, by the same name, was independently defined by Dolev and Rajsbaum. Sometimes called *Long-lived consensus*, it concerns maintaining consensus on a single bit in a self-stabilizing fashion [6].

[3] In [11] we defined both non-uniform and a uniform variants of continuous consensus. In the (m,t) model processes can fail and recover repeatedly, so we focus on the uniform variant, in which *all* processes maintain the same core at all times.

can immediately be implemented, but so can, say, consistent resource allocation, mutual exclusion, etc. In a world in which distributed systems increasingly interact with an outside world, a CC protocol facilitates the system's ability to present a consistent view to the world.

The consensus problem has rightfully attracted considerable attention in the last thirty years, since it is a basic primitive without which many other tasks are unattainable. Continuous consensus offers a strict generalization of (simultaneous) consensus: While consensus is concerned with agreeing on a single bit, continuous consensus allows decisions to be taken based on a broader picture involving a number of events of interest. In particular, it provides the processes with Byzantine Agreement regarding the values of all relevant external inputs that do or do not arrive at the various processes (where relevance is determined by $\mathcal{E}$). It thus eliminates the need for initiating a separate instance of a consensus protocols for each of the facts of interest, and provides the same benefits at a lower cost.

Popular failure assumptions bound the overall number of failures that may occur during the execution of a protocol [17]. Clearly, if the adversary can cause all of the processes to fail (by crashing, say) then the best protocols cannot be expected to achieve much. Typically, a process is considered *faulty* in a given run if it ever displays incorrect behavior during the course of the run. Such assumptions are reasonable for applications that are short-lived. For applications such as continuous consensus, however, which is an ongoing service that should operate indefinitely, expecting (or assuming) that certain processes remain correct throughout the lifetime of the system may be overly optimistic. Conversely, it would also be natural to expect various failed processes to be repaired over time, and thus resume correct participation in the protocol. In such a setting, it is more reasonable to consider bounds on the number and/or types of failures that can occur over limited intervals of time. Indeed, Castro and Liskov designed a protocol for state-machine replication in an Byzantine environment that is correct provided that no more than $n/3$ processes fail during an execution-dependent period that they call a *window of vulnerability* [2]. We follow a similar path in this paper, and consider continuous consensus in omissive settings under the (m,t)-*interval bound* assumption (called the (m,t) *model* for short), which states that there is no m-round interval in which more than t processes display faulty behavior. The contributions of this paper are:

- The (m,t)-interval bounded omission failure model is introduced.
- A continuous consensus protocol mt-CC for the (m,t) model whenever $t < m$ is presented.
- mt-CC is shown to be optimal *in all runs* for this model: For any CC protocol P and any given behavior of the adversary, the core maintained by mt-CC at each time k is a superset of the core maintained by P under the same conditions.
- The lower bound used in proving the optimality of mt-CC is the first one tackling the possibility of process recoveries in a simultaneous agreement problem. All previous lower bounds for simultaneous consensus and continuous consensus in the presence of crash or omission failures make essential use of the fact that failures accumulate monotonically over time [7, 11, 12, 13] The new lower bound proof in this paper overcomes this hurdle.
- The main lemmas in the lower bound apply to a large class of benign failure models, including the general omissions model studied in [13].

2 Model and Preliminary Definitions

Our treatment of the continuous consensus problem will be driven by a knowledge-based analysis. A general approach to modeling knowledge in distributed systems was initiated in [9] and given a detailed foundation in [8] (most relevant to the current paper are Chapters 4 and 6). For ease of exposition, our definitions will be tailored to the proofs for continuous consensus in our setting.

The Communication Network. We consider a synchronous network with $n \geq 2$ possibly unreliable processes, denoted by $\mathbb{P} = \{1, 2, \ldots, n\}$. Each pair of processes is connected by a two-way communication link. Processes correctly identify the sender of every message they receive. They share a discrete global clock that starts out at time 0 and advances by increments of one. Communication in the system proceeds in a sequence of *rounds*, with round $k+1$ taking place between time k and time $k+1$. Each process starts in some *initial state* at time 0. Then, in every following round, the process first sends a set of messages to other processes, and then receives messages sent to it by other processes during the same round. In addition, a process may also receive requests for service from clients external to the system (think, for example, of deposits and withdrawals at branches of a bank), or input from sensors with information about the world outside of the system (e.g., smoke detectors). Finally, the process may perform local computations based on the messages it has received. The history of an infinite execution of such a network will be called a *run*.

Modeling the Environment: Inputs and Failures. A protocol is designed to satisfy a specification when executed within a given setting, which determines the aspects of the execution that are not controlled by the protocol. In our framework the setting can be described in terms of a set of adversaries that control the two central aspects of any given run: inputs and failures.

Inputs. Every process starts out in an initial local state from some set Σ_i, and can receive an external input in any given round k (this input is considered as *arriving* at time k). The initial local state of each process can be thought of as its external input at time 0. We represent the external inputs in an infinite execution as follows. Define a set $\mathbb{V} = \mathbb{P} \times \mathbb{N}$ of *process-time nodes* (or *nodes*, for short). We shall use a node $(i,k) \in \mathbb{V}$ to refer to process i at time k. We denote by $\mathbb{V}(k)$ the set $\mathbb{P} \times \{k\} \subset \mathbb{V}$ of all nodes (i,k), and if $k \leq \ell$ then we define $\mathbb{V}[k,\ell] = \mathbb{P} \times \{k, k+1, \ldots, \ell\} = \mathbb{V}(k) \cup \cdots \cup \mathbb{V}(\ell)$.

An *(external) input assignment* is a function ζ associating with every node $(i,0)$ at time 0 an initial state from Σ_i and with each node (i,k) with $k > 0$ an input from a set of possible inputs, which is denoted by $\mathbb{I}$. (The set $\mathbb{I}$ typically contains a special symbol $\perp$, corresponding to a "null" external input.) An *input model* consists of a set Ξ of input assignments. For the purpose of the analysis in this paper, we will focus on input models in which the inputs at different nodes of $\mathbb{V}$ are not correlated. An input model Ξ is said to be *independent* if for every $\zeta, \zeta' \in \Xi$ and every set $T \subseteq \mathbb{V}$, we are guaranteed that $\zeta_T \in \Xi$, where ζ_T is the input assignment that coincides with ζ on T and with ζ' on $\mathbb{V} \setminus T$. In the classical consensus problem, for example, $\Sigma_i = \{0,1\}$ and $I = \{\perp\}$. The input model consists of all possible input assignments based on these Σ_i and on I, and is clearly independent.

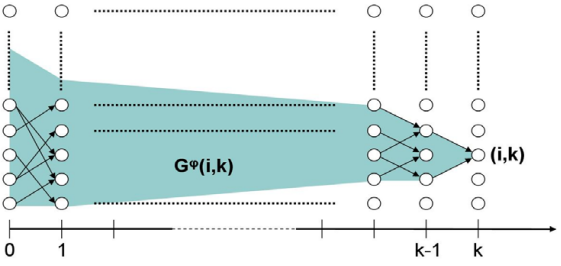

Fig. 1. A communication graph G^φ and i's subgraph $\mathsf{G}^\varphi(i,k)$

Benign failures and recoveries. The second aspect of a run that is determined by the adversary has to do with the type of failures possible, and the patterns in which they can occur. When focusing on benign failures, as in this paper, any message that is delivered is sent according to the protocol.

A *failure pattern* for benign failures is a function $\varphi : \mathbb{V} \to 2^{\mathbb{P}}$. Intuitively, a failure pattern determines which channels are blocked (due to failure) in any given round. More concretely, $\varphi(i,k)$ is the set of processes to which messages sent by i in round $k+1$ will not be delivered.[4] Equivalently, a message sent by i to j in round $k+1$ will be delivered iff $j \notin \varphi(i,k)$. We denote by $\ddot\smile$ the failure-free pattern satisfying that $\ddot\smile(v) = \emptyset$ for all $v \in \mathbb{V}$.

We identify a failure pattern φ with a *communication graph*,[5] denoted by G^φ, detailing the active channels in an execution. We define the graph $\mathsf{G}^\varphi = (\mathbb{V}, E^\varphi)$, where $E^\varphi = \{\langle (i,k),(j,k+1)\rangle : j \notin \varphi(i,k)\}$. Notice that φ uniquely determines G^φ and vice-versa. For a node $v = (i,k) \in \mathbb{V}$, we denote by $\mathsf{G}^\varphi(i,k)$ (or $\mathsf{G}^\varphi(v)$) the subgraph of G^φ generated by v and all nodes $w \in \mathbb{V}$ such that there is a directed path from w to v in G^φ. This subgraph captures the potential "causal past" of v under φ: all nodes by which v can be affected via communication, either directly or indirectly. An illustration of a graph $\mathsf{G}^\varphi(i,k)$ is depicted in Figure 1. Given a set of nodes $S \subseteq \mathbb{V}$, we denote by $\mathsf{G}^\varphi(S)$ the subgraph of G^φ obtained by taking the union of the graphs $\mathsf{G}^\varphi(v)$, for $v \in S$.

A *(benign) failure model* for synchronous systems is a set Φ of failure patterns φ. Intuitively we can view a benign failure model as one in which failures affect message deliveries. Any message that *is* delivered is one that was sent according to the protocol. The standard benign models, including t-bounded crash failures, sending omissions, receiving omissions, or general omissions [13] are easily modeled in this fashion. The same applies to models that bound the number of undelivered messages per round (e.g., [3, 18]).

[4] Note that a failure pattern can model process failures as well as communication failures. If $i \in \varphi(i,k)$ then i "does not receive its own message" in round $k+1$. This corresponds to a loss of i's local memory, and captures the *recovery from a process crash*. It allows a distinction between recovery from a local process crash and reconnection following disconnected operation.

[5] Communication graphs were first used, informally, in the anasysis of consensus by Merritt [10]. They were formalized in [8, 13]. Our modelling is taken from the latter. Similar modelling was more recently used in the Heard-Of model of Charron-Bost and Schipper [3].

The (m,t) interval-bounded failure model. For the purpose of this paper, it is convenient to consider a process $i \in \mathbb{P}$ as being *faulty at* (i,k) *according to* φ if $\varphi(i,k) \neq \emptyset$ (i.e., if one or more of i's outgoing channels is blocked in round $k+1$). We say that $i \in \mathbb{P}$ is *faulty according to* φ in the interval $\mathbb{V}[k,k+m-1]$ if i is faulty at (i,k') for some k' satisfying $k \leq k' \leq k+m-1$. The (m,t) interval-bounded failure model is captured by the set $\mathrm{OM}(m,t)$ of all failure patterns φ such that, for every $k \geq 0$, no more than t processes are faulty according to φ in $\mathbb{V}[k,k+m-1]$. The standard (sending) omissions failure model is then captured by $\mathrm{OM}(\infty,t) = \bigcap_{m \geq 1} \mathrm{OM}(m,t)$.

Environments and Protocols. We shall design protocols for continuous consensus with respect to an environment that is specified by a pair $\Delta = \Phi \times \Xi$, where $\Phi = \mathrm{OM}(m,t)$ and Ξ is an independent input model. Each element (φ,ζ) of the environment Δ we call an *adversary*. A protocol for process i is a deterministic function P_i from the local state of a process to local actions and an assignment of at most one message to each of its outgoing links. Its local state at time $k+1$ consists of its local variables after the local actions at time k are performed, its external input at $(i,k+1)$, and the messages delivered to it in round $k+1$ on its incoming channels.[6]

A *joint protocol* (or just protocol for short) is a tuple $P = \{P_i\}_{i \in \mathbb{P}}$. For a given adversary $(\varphi,\zeta) \in \Phi \times \Xi$, a protocol P determines a unique *run* $r = P(\varphi,\zeta) : \mathbb{V} \to L \times M \times \mathbb{P}$, which assigns a local state to each node (i,k) in the unique manner consistent with P, φ and ζ: initial states and external inputs are determined by ζ, protocol P determines local states and outgoing messages, and φ determines which of the sent messages on incoming channels produces a delivered message.

3 Continuous Consensus and Common Knowledge

Knowledge theory, and specifically the notion of common knowledge, are central to the study of simultaneously coordinated actions. This connection has been developed and described in [7, 8, 11, 13, 15, 16]. In particular, [11] showed that in a continuous consensus protocol, the contents of the core at a given time k are guaranteed to be common knowledge. In this section we will review this connection, in a manner that will be light on details and focus on the elements needed for our analysis of CC protocols. In particular, we will make use of a very lean logical language in which the only modal construct is common knowledge. For a more complete exposition of knowledge theory see [8].

We are interested in studying CC in an environment $\Delta = (\Phi,\Xi)$ in which Ξ is an independent input model. We say that Ξ is *non-degenerate* at a node $(i,k) \in \mathbb{V}$ if there are $\zeta,\zeta' \in \Xi$ such that $\zeta(i,k) \neq \zeta'(i,k)$. A CC application needs to collect information only about non-degenerate nodes, since on the remaining nodes the external input is fixed a priori. We will consider a set of *primitive events* $\mathcal{E}$ w.r.t. Ξ each of the form $e = \langle \alpha,i,k \rangle$ where $\alpha \in \Xi$ and Ξ non-degenerate at (i,k). The event $e = \langle \alpha,i,k \rangle$ is said to occur in run $r = P(\varphi,\zeta)$ exactly if $\zeta(i,k) = \alpha$. The core will be assumed to consist of a set of such primitive events.

[6] To model local crashes, we would replace the local variables in i's local state at time k by λ if $i \in \varphi(i,k+1)$.

Knowledge is analyzed within the context of a *system* $\mathcal{R} = \mathcal{R}(P,\Delta)$ consisting of the set of runs $r(P,\varphi,\zeta)$ of a protocol P in Δ. A pair (r,k) where r is a run and k is a time is called a *point*. Process i's local state at (i,k) in r is denoted by $r_i(k)$. We say that i *cannot distinguish* (r,k) from (r',k) if $r_i(k) = r'_i(k)$. We consider knowledge formulas to be true or false at a given point (r,k) in the context of a system $\mathcal{R}$. In the standard definition [8], process i knows a fact A at (r,k) if A holds at all points that i cannot distinguish from (r,k).

For every $e \in \mathcal{E}$, we denote by $C(e)$ the fact that e is common knowledge, which intuitively means that everybody knows e, everybody knows that everybody knows e, and so on ad infinitum. We define common knowledge formally in the following way, which can be shown to capture this intuition in a precise sense. We define a *reachability relation* $\sim$ among points of $\mathcal{R}$ to be the least relation satisfying:

1. if $r_i(k) = r'_i(k)$ then $(r,k) \sim (r',k)$, and
2. if, for some $r'' \in \mathcal{R}$, both $(r,k) \sim (r'',k)$ and $(r'',k) \sim (r',k)$, then $(r,k) \sim (r',k)$.

In other words, define the *similarity graph* over $\mathcal{R}$ to be an undirected graph whose nodes are the points of $\mathcal{R}$, and where two points are connected by an edge if there is a process that cannot distinguish between them. Then $(r,k) \sim (r',k)$ if both points are in the same *connected component* of the similarity graph over $\mathcal{R}$. Notice that $\sim$ is an equivalence relation, since connected components define a partition on the nodes of an undirected graph. We denote by $(\mathcal{R},r,k) \models C(e)$ the fact that e is common knowledge to the processes at time k in r. We formally define:

$$(\mathcal{R},r,k) \models C(e) \quad \text{if event } e \text{ occurs in every } r' \in \mathcal{R} \text{ satisfying } (r,k) \sim (r',k).$$

We can formally prove that the events in the core of a CC protocol must be common knowledge:

Theorem 1 ([7, 11]). *Let P be a CC protocol for Δ, and let $\mathcal{R} = \mathcal{R}(P,\Delta)$. For all runs $r \in \mathcal{R}$, times $k \geq 0$ and events $e \in \mathcal{E}$, we have:*

$$\textit{If} \quad e \in M_i^r[k] \quad \textit{then} \quad (\mathcal{R},r,k) \models C(e).$$

4 Lower Bounds for CC in Benign Failure Models

We can show that an event at a node (i,k) cannot be in the the core of a CC protocol at time ℓ if we prove that the event is not common knowledge by time ℓ. It turns out that the failure models and failure patterns play a central role in forcing events *not* to be common knowledge. By working with these directly, we avoid some unnecessary notational clutter.

Given a failure model Φ, we define the *similarity* relation $\approx$ on $\Phi \times \mathbb{N}$ to be the least relation satisfying:

1. if $\mathsf{G}^{\varphi}(i,k) = \mathsf{G}^{\varphi'}(i,k)$ holds for some process $i \in \mathbb{P}$, then $(\varphi,k) \approx (\varphi',k)$, and
2. if, for some $\varphi'' \in \Phi$, both $(\varphi,k) \approx (\varphi'',k)$ and $(\varphi'',k) \approx (\varphi',k)$, then $(\varphi,k) \approx (\varphi',k)$.

As in the case of $\sim$, the $\approx$ relation is an equivalence relation, because condition (1) is reflexive and symmetric, while condition (2) is symmetric and guarantees transitivity.

We say that process i is *shut out* in round $k+1$ by φ (equivalently, (i,k) is shut out by φ), if $\varphi(i,k) \supseteq \mathbb{P} \setminus \{i\}$. Intuitively, this means that no process receives a message from i in round $k+1$. We say that a node (i,k) is *hidden by* φ *at time* ℓ if $(\varphi,\ell) \approx (\psi,\ell)$ for some pattern ψ in which i is shut out rounds $k+1,\ldots,\ell$. If a node is hidden by the failure pattern, then its contents cannot be common knowledge, no matter what protocol is being used, as formalized by the following theorem:

Theorem 2. *Let $\mathcal{R} = \mathcal{R}(P,\Delta)$, where $\Delta = \Phi \times \Xi$ and Ξ is independent, let $e = (\alpha,i,k)$ be a primitive event and let $r = P(\varphi,\zeta) \in \mathcal{R}$. If (i,k) is hidden by φ at ℓ then $(\mathcal{R},r,\ell) \not\models C(e)$.*

A major tool in our impossibility results is the notion of a covered set of nodes. Intuitively, a covered set S at a given point satisfies three main properties. (i) it is not common knowledge that even on node of s is faulty, as there is a reachable point in which all nodes if S are nonfaulty. (ii) for every node in s, its contents are not common knowledge, in the sense that there is a reachable point in which this node is silent. (iii) Finally, the reachable points in (i) and (ii) all agree with the current point on the nodes not in S. Formally, we proceed as follows.

Definition 1 (Covered Nodes). *Let $S \subset \mathbb{V}$, and denote $\bar{S}_\ell = \mathbb{V}[0,\ell] \setminus S$. We say that S is* covered *by φ at time ℓ (w.r.t. Φ) if*

- *for every node (i,k)=$v \in S$ there exist φ_v such that (a) $(\varphi,\ell) \approx (\varphi_v,\ell)$, (b) $\mathsf{G}^\varphi(\bar{S}_\ell) = \mathsf{G}^{\varphi_v}(\bar{S}_\ell)$, and (c) i is shut out in rounds $k+1,\ldots,\ell$ of φ_v. Moreover,*
- *there exists $\varphi' \in \Phi$ such that (d) $(\varphi,\ell) \approx (\varphi',\ell)$, (e) $\mathsf{G}^\varphi(\bar{S}_\ell) = \mathsf{G}^{\varphi'}(\bar{S}_\ell)$, and (f) $\varphi'(v) = \emptyset$ for all nodes $v \in S$.*

The fact that $\approx$ is an equivalence relation immediately yields

Lemma 1. *Fix Φ and let $S \subseteq \mathbb{V}$. If $(\varphi,\ell) \approx (\varphi',\ell)$ and $\mathsf{G}^\varphi(\bar{S}_\ell) = \mathsf{G}^{\varphi'}(\bar{S}_\ell)$ then S is covered by φ at ℓ iff S is covered by φ' at ℓ.*

One set that is guaranteed to be covered at time ℓ is $\mathbb{V}(\ell)$:

Lemma 2. *$\mathbb{V}(\ell)$ is covered by φ at ℓ, for every failure pattern φ and $\ell \geq 0$.*

Proof. Denote $\mathbb{V}(\ell)$ by S. Choose $\varphi = \varphi_v$ for each $v = (j,\ell) \in S$ to satisfy clauses (a), (b) and (c) of the definition for $v \in S$. The clauses are immediate for φ_v. To complete the claim we need to show that S is covered by φ at ℓ: Define φ' for clauses (d), (e) and (f) to be the pattern obtained from φ by setting $\varphi'(j,\ell) = \emptyset$ for every $j \in \mathbb{P}$. ■

Monotonicity. Our "lower bound" proofs take the form of proving impossibility of common knowledge under various circumstances. To make the results in this section widely applicable, we state and prove results with respect to classes of failure models rather than just to the (m,t) model. We shall focus on models with the property that reducing the set of blocked edges in a failure pattern yields a legal failure pattern. Formally, we say that ψ *improves on* φ, and write $\psi \sqsubseteq \varphi$, if $\psi(v) \subseteq \varphi(v)$ for every $v \in \mathbb{V}$. (Notice that

the fault-free failure pattern ☺ satisfies ☺ $\sqsubseteq \varphi$ for all patterns φ.) It is easy to check that $\psi \sqsubseteq \varphi$ iff G^{φ} is a subgraph of G^{ψ}. A failure model Φ is *monotonically closed* (or *monotone*, for short) if for every pattern ψ that improves on a pattern in Φ is itself in φ. Formally, $\varphi \in \Phi$ and $\psi \sqsubseteq \varphi$ implies $\psi \in \Phi$. Clearly, if Φ is monotonically closed, then ☺ $\in \Phi$. Observe that $\textsc{Om}(m,t)$ is monotonically closed.

Single Stalling. In many failure models, a crucial obstacle to consensus is caused by executions in which a single process is shut out in every round. This will also underly the following development, where we show essentially that for a broad class of failure models, the adversary's ability to fail an arbitrary process per round will keep any events from entering the CC core. We call this *single stalling*. To capture this formally for general failure models, we make the following definitions.

We say that two patterns φ and φ' *agree* on a set of nodes $T \subseteq \mathbb{V}$ if $\mathsf{G}^{\varphi}(T) = \mathsf{G}^{\varphi'}(T)$. Notice that this neither implies or is implied by having $\varphi(v) = \varphi'(v)$ for all $v \in T$. This notion of agreement depends on the nodes with directed paths into nodes $v \in T$, while $\varphi(v)$ specifies the outgoing edges from v. In a precise sense, though, the execution of a given protocol P on failure pattern φ can be shown to be indistinguishable at the nodes of T from its execution on φ'. This is closely related to the the structure underlying the proof of Theorem 2.

Definition 2 (Single stalling). *Let Φ be a monotone failure model. Fix Φ, let $\varphi \in \Phi$, and let $W \subset \mathbb{V}$. We say that $\mathsf{G}^{\varphi}(W)$* admits single stalling in $[k,\ell]$ *(w.r.t. Φ) if, for every sequence $\sigma = p_{k+1}, \ldots, p_{\ell}$ of processes (possibly with repetitions), there exists a pattern $\varphi_\sigma \in \Phi$ agreeing with φ on W such that both (a) $(\varphi,\ell) \approx (\varphi_\sigma,\ell)$, and (b) each process p_j in σ is shut out in round j of in φ_σ, for all $j = k+1,\ldots,\ell$.*

The heart of our lower bound proof comes in the following lemma. Roughly speaking, it states that if a set of nodes S containing the block $\mathbb{V}[k+1,\ell]$ of nodes from time $k+1$ to ℓ are all covered, and it is consistent with the information stored in the complement set $\bar{S}_\ell$ that the node (j,k) could be shut out, then (j,k) can be added to the set of covered nodes. to the set of covered nodes. This allows to prove incrementally that the nodes at and after the critical time $c = c(\varphi)$ that are not in the critical set are all covered. Formally, we now show the following:

Lemma 3. *Fix a monotone failure model Φ. Let $k < \ell$ and assume that the set $S \supseteq \mathbb{V}[k+1,\ell]$ is covered by φ at ℓ. Let $T = S \cup \{(j,k)\}$, and let ψ be the pattern obtained from φ by shutting out (j,k). If $\psi \in \Phi$ and $\mathsf{G}^{\psi}(\bar{T}_\ell)$ admits single stalling in $[k+1,\ell]$, then T is hidden by φ at ℓ.*

The proof of Lemma 3 appears in the Appendix. Based on Lemma 3, we obtain:

Lemma 4. *Let $k < \ell$. If $S = \mathbb{V}[k+1,\ell]$ is covered by φ at ℓ and $\mathsf{G}^{\varphi}(\bar{S}_\ell)$ admits single stalling in $[k,\ell]$, then $\mathbb{V}[k,\ell]$ is covered by φ at ℓ.*

A new lower-bound construction. Previous lower bounds on common knowledge in the presence of failures, including the ones for CC protocols in [11], are all based on the fixed-point construction of Moses and Tuttle [13] and, for uniform consensus, on its extension by Neiger and Tuttle [16]. Their construction applies to crash and sending

omissions, and it (as well as earlier proofs in [7]) depends in an essential way on the fact that faultiness in these models is forever. Somewhat in the spirit of the "Heard-of" model of [3], our generic lower bound results in Lemmas 2–4 are oblivious of the notion of process failures per-se. Lemmas 3 and 4 are the basis of our new a fixed-point construction for general monotone failure models that subsumes the construction in [13].

5 A Continuous Consensus Protocol for the (m,t) Model

The purpose of this section is to present mt-CC, a CC protocol for $\text{OM}(m,t)$ that is efficient and optimal in runs. The protocol makes use of instances of the UniConCon protocol (UCC) presented in [11]. This is an optimal (uniform) CC protocol for $\text{OM}(\infty,t)$ with the following properties: Beyond storing the relevant events in $\mathcal{E}$—which are application dependent—UCC uses $O(\log k)$ new bits of storage and performs $O(n)$ computation in each round k. In addition, in every round $k+1$ each process i sends everyone a message specifying the set $f_i[k]$ of processes it knows are faulty at time k, as well as any new information about events $e \in \mathcal{E}$ that it has obtained in the latest round. The failure information in every message requires $\min\{O(n), O(|f_i[k]|\log n)\}$ bits. Our analysis will not need further details about UCC beyond these facts.

The intuition behind our protocol is based on the following observation. The adversary's ability to cause failures in the $\text{OM}(m,t)$ model in a given round depends only on the failures that the culprit caused in the m-round interval ending with this round. In a precise sense, the (crucial) impact of the adversary's actions on the core at time ℓ depends only on the previous cores and the failures that occur in the last m rounds. Consequently, our strategy is to invoke a new instance of UCC in every round, and keep it running for m rounds. Each instance takes into account only failures that occur after it is invoked. For times $\ell \leq m$, the core $\hat{M}[\ell]$ coincides with the one computed by the UCC initiated at time 0. For later times $\ell > m$, the core $\hat{M}[\ell]$ is obtained by taking the union of $\hat{M}[\ell-1]$ and the core $M[\ell]_{\ell-m}$ obtained by the UCC instance invoked at time $\ell - m$.[7]

The mt-CC protocol shown in Figure 2 presents the mt-CC protocol for CC in $\text{OM}(m, t)$. It accepts m and $t < n$ as parameters. We denote by $M[k]_s$ the core computed by $\text{UCC}_s(t)$ at time k. The message sending operations in the instances UCC_s are suppressed, replaced by the message μ sent by mt-CC. More formally, UCC_s is an instance of UCC that is invoked at time s, in which only failures that occur from round $s+1$ on are counted. It is initiated with no failure information. Incoming failure information for UCC_s is simulated based on the incoming messages μ, where only failures that occur in rounds $s+1\ldots,s+m$ are taken into account. Similarly, maintaining the events in the core is handled by mt-CC. Based on the structure of UCC it is straightforward to show:

Lemma 5. *Let $\varphi,\varphi' \in \text{OM}(m,t)$ be failure patterns, such that no process fails in the first s rounds of φ', and the same channels are blocked in both patterns from round $s+1$ on. Let r be a run of UCC_0 with adversary (φ',ζ), and let r' be a run with adversary (φ,ζ) in which UCC_s is invoked at time s. Then $M_x[s+m]_s = M'_x[s+m]$ for all $x \in \mathbb{P}$.*

[7] In fact, the UCC instance computes the subset of nodes of $\mathbb{V}[0,\ell]$ that determines its core, and the core $M[\ell]_{\ell-m}$ consists of the events that occur at the nodes of this subset.

```
mt-CC_x        % Executed by x, on parameters m and t
0  M̂_x[0] ← ∅
1  invoke an instance of UCC_0
   for every round k ≥ 1  do
2     send μ = ⟨failure-info, new-events⟩ to all
3     s ← max(0, k − m)
4     receive incoming messages;
5     simulate a round of UCC_s, ..., UCC_{k−1};
6     M̂_x[k] ← (M̂_x[k − 1] ∪ M_x[k]_s)
7     invoke an instance of UCC_k
   endfor
```

Fig. 2. Process x's computation in mt-CC

The `failure-info` component of the message μ on line 2 consists of a list of the processes j that x knows have displayed faulty behavior in the last m rounds, and for each such j the latest round number (mod m) in which x knows j to have been faulty. In mt-CC at most m instances of UCC_k need to be active at any given time. As we show in the full paper, simple accounting mod m suffices for x to be able to construct the incoming (simulated) messages for each active instance, based on the `failure-info` components of incoming messages. The failure component in mt-CC message is thus of size $\min\{O(n\log m), O(|f_i[k]|\cdot\log nm)\}$ bits, where now $f_i[k]$ is the number of processes known by i to have failed after round $k-m$. The `new-events` component in mt-CC messages is the same as in a single instance of UCC in the standard sending omissions model $\text{OM}(\infty,t)$.

While mt-CC appears to be rather straightforward, the use of a *uniform* CC protocol to construct it, and the way in which the results of the different instances are combined is rather subtle. The real technical challenge, which brought about the new lower bound techniques in Section 4, is proving that it is optimal in all runs. We now consider the properties of mt-CC. Our first result is an upper bound, which is proved in Section B of the appendix:

Lemma 6. *The mt-*$\text{CC}(m,t)$ *is a CC protocol for* $m > t$. *When* $m \leq t$ *it satisfies Accuracy and Consistency.*

Based on the lower bound in Section 4 we then prove that mt-CC is optimal for $m > t$. Moreover, the results of Santoro and Widmayer can be used to show that no CC protocol exists for $m \leq t$ (this also easily follows from our lower bounds). Nevertheless, mt-CC is optimal among the protocols that satisfy accuracy and consistency for $m \leq t$. The optimality property implies that, essentially, mt-CC is as complete as possible for $m \leq t$.

Being based on the cores computed by instances of UCC, the core at time ℓ in a run of mt-CC is the set of events that take place at a particular set $T \subset \mathbb{V}[0,\ell]$. Moreover, since UCC_s in $\text{OM}(m,t)$ is equivalent to UCC in the standard sending omissions model (by Lemma 5) the set A has the same structure as derived in the Moses and Tuttle construction. Let c denote the maximal time in nodes of T. To prove that mt-CC is optimal, we show that all nodes that are at and after the critical time $c = c(\varphi,\ell)$ and are not in A are covered. Formally, we apply Lemma 4 to mt-CC in the $\text{OM}(m,t)$ model to obtain:

Lemma 7. *Let r be a run of mt-CC(m,t) with failure pattern φ, and assume that $\hat{M}[\ell]$ is generated by the set of nodes $A \subseteq \mathbb{V}[0,\ell]$. If c is the maximal time of any node in A, then $S = (\mathbb{V}[c,\ell] \setminus A)$ is covered by φ at ℓ.*

In the OM(m,t) model, if a process fails to deliver a message in round k, it is faulty. Hence, all nodes of $\mathbb{V}[0,\ell-1]$ that do not appear in the causal past of A belong to faulty processes. We can use Lemma 7 to silence them at a reachable point using the covered nodes in the same way as in the proof on Lemma 3. Once we do this, all nodes from before time ℓ that do not appear in the view $\mathsf{G}^{\varphi}(A)$ are eliminated from the graph. Formally, we can show:

Lemma 8. *Let r be a run of mt-CC(m,t) and let ℓ and A be as in Lemma 7. Then all nodes in $\mathbb{V}[0,\ell] \setminus A$ are hidden by φ at ℓ in* OM(m,t).

Lemma 8 provides the final ingredient for the optimality proof for mt-CC: Lemma 6 guarantees that mt-CC solves continuous consensus. Lemma 8 states that all nodes not in the core view $\mathsf{G}^{\varphi}(A)$ of mt-CC are hidden given the current failure pattern, for all protocols. Theorem 2 states that an event at a hidden node cannot be common knowledge, which by Theorem 1 implies that it cannot be contained in the common core under *any* protocol whatsoever. It follows that, for each and every adversary all times ℓ, the core provided by mt-CC is a superset of the core that any other correct CC protocol can construct. We obtain:

Theorem 3. *Let $m > t \geq 0$, and let $\Delta =$* OM$(m,t) \times \Xi$ *where Ξ is independent. Then mt-CC(m,t) is optimal in all runs in Δ.*

We can also show that mt-CC provides optimal behavior for $m \leq t$; it is accurate, consistent and maximal. Indeed, for $t \geq m > 1$, there are runs of mt-CC in which nontrivial events enter the core. However, the completeness property is not achievable in some of the runs. Using Lemmas 4 and 3 we can show:

Lemma 9. *Let $0 < m \leq t$, and let $\Delta =$* OM$(m,t) \times \Xi$ *where Ξ is independent. Let r be a run of a CC protocol with failure pattern $\varphi \in$* OM(m,t) *in which no more than one process fails per round in rounds $1,\ldots,\ell$. Then the core at time ℓ is necessarily empty.*

By Lemma 9, as well as from the results of Santoro and Widmayer in [18], it follows that Continuous consensus is not solvable in OM(m,t) for $1 \leq m \leq t$. Lemma 9 implies that if $m = t = 1$ then the core is necessarily empty at all times. However, whenever $m > 1$ there are runs in which the core is *not* empty. In these cases, Lemma 6 guarantees that mt-CC is Accurate and Consistent. In fact, it is as close to a CC protocol as one could hope for:

Theorem 4. *For all $m \leq t$, the mt-CC protocol is optimal in all runs among the protocols that satisfy the Accuracy and Consistency properties of CC.*

6 Conclusions

Continuous consensus (CC) is a powerful primitive in synchronous distributed systems. Being an ongoing activity, the classical t-bounded failure model in which failures can

only accumulate and recoveries are not considered is not satisfactory. This paper considers the CC problem in an (m,t) omission failure model. mt-CC, an efficient protocol for CC in such models is presented, and is shown to be a optimal in all runs: It maintains the largest core at all times, in each and every run (i.e., against each behavior of the adversary). We remark that while this paper focused on optimally fast CC protocols, it is an interesting open problem how to trade speed against message complexity in CC protocols.

The lower bound proof makes essential use of the theory of knowledge in distributed systems. Using a new technique the lower bound manages to sidestep previous dependence on the stability of faultiness in order to apply to models with failures and recoveries, such as the (m,t) omission model. It gives rise to a lower-bound construction generalizing that of Moses and Tuttle [13] in the standard sending omissions model. The fact that mt-CC is optimal in all runs proves that, in fact, the construction completely characterizes what is common knowledge (and what can appear in a CC core) in $\text{OM}(m,t)$. The new construction applies to monotone failure models in general. It thus strictly generalizes the MT construction and applies to many other failure models, including the elusive general omissions failure model [13] in which a faulty process can fail to send *and* to receive messages. Models that bound the number of messages lost are also monotone, as are ones in which there are different reliability guarantees for different subsets of the processes (say central servers vs. plain workstations). We believe that a slight variation on this construction can be used to solve a twenty-year old open problem regarding optimum simultaneous consensus in the general omissions model. Finally, in future work we plan to show that, with slight modifications, the new lower bound proof is also applicable to topologies in which the communication network is not a complete graph.

References

1. Burns, J.E., Lynch, N.A.: The byzantine firing squad problem. Technical Report MIT/LCS/TM-275 (1985)
2. Castro, M., Liskov, B.: Proactive recovery in a Byzantine-fault-tolerant system. In: Proc. 4th OSDI: Symp. Op. Sys. Design and Implementation, pp. 273–288 (2000)
3. Charron-Bost, B., Schiper, A.: The Heard-Of Model: Unifying all Benign Failures. EPFL LSR-REPORT-2006-004 (2006)
4. Coan, B.A., Dolev, D., Dwork, C., Stockmeyer, L.J.: The distributed firing squad problem. SIAM J. Comput. 18(5), 990–1012 (1989)
5. Dolev, D., Reischuk, R., Strong, H.R.: Eventual is earlier than immediate. In: Proc. 23rd IEEE Symp. on Foundations of Computer Science, pp. 196–203 (1982)
6. Dolev, S., Rajsbaum, S.: Stability of long-lived consensus. J. Comput. Syst. Sci. 67(1), 26–45 (2003)
7. Dwork, C., Moses, Y.: Knowledge and common knowledge in a Byzantine environment: crash failures. Information and Computation 88(2), 156–186 (1990)
8. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge (1995) (revised 2003)
9. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. Journal of the ACM 37(3), 549–587 (1990)

10. Merritt, M.J.: Unpublished notes on the Dolev-Strong lower bound for Byzantine Agreement (1984)
11. Mizrahi, T., Moses, Y.: Continuous consensus via common knowledge. Distributed Computing 20(5), 305–321 (2008)
12. Moses, Y., Raynal, M.: Revisiting Simultaneous Consensus with Crash Failures. Tech Report 1885, 17 pages, IRISA, Université de Rennes 1, France (2008), http://hal.inria.fr/inria-00260643/en/
13. Moses, Y., Tuttle, M.R.: Programming simultaneous actions using common knowledge. Algorithmica 3, 121–169 (1988)
14. Mostéfaoui, A., Rajsbaum, S., Raynal, M.: Synchronous condition-based consensus. Distributed Computing 18(5), 325–343 (2006)
15. Neiger, G., Bazzi, R.A.: Using knowledge to optimally achieve coordination in distributed systems. Theor. Comput. Sci. 220(1), 31–65 (1999)
16. Neiger, G., Tuttle, M.R.: Common knowledge and consistent simultaneous coordination. Distributed Computing 6(3), 181–192 (1993)
17. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (1980)
18. Santoro, N., Widmayer, P.: Time is not a healer. In: Proc. 6th Symp. Theo. Asp. Comp. Sci (STACS), pp. 304–313 (1989)

A Lower Bound Proof

This section contains the proof of our central lower bound claim.

Proof of Lemma 3: Let k, ℓ, φ, ψ, S and $T = S \cup \{(j,k)\}$ satisfy the conditions of the lemma. Since S is covered by φ at ℓ, we have that $(\varphi,\ell) \approx (\varphi',\ell)$ where φ' agrees with φ on $\bar{S}_\ell$, and in which all nodes of S are correct in φ'. For ease of exposition we assume w.l.o.g. that all nodes of S are correct in φ.

Denote by φ_m the pattern that agrees with φ on $V \setminus \{(j,k)\}$, where $\varphi_m(j,k)=\varphi(j,k) \cup \{1,\ldots,m\}$. In particular, we vacuously have $\varphi_0 = \varphi$. Observe that $\varphi \sqsubseteq \varphi_m \sqsubseteq \psi$ holds for all m. We prove by induction on m that $(\varphi,\ell) \approx (\varphi_m,\ell)$. The claim trivially holds for $m = 0$. Let $m = h+1 > 0$ and assume inductively that the claim holds for h. Thus, $(\varphi,\ell) \approx (\varphi_h,\ell)$. By Lemma 1 it follows that $S = V[k+1,\ell]$ is covered at φ_h. Denote $w = (h+1,k+1)$. Clearly, $w \in S$. Let φ_w be a failure pattern such that $(\varphi_h,\ell) \approx (\varphi_w,\ell)$, $\mathsf{G}^{\varphi_h}(\bar{S}_\ell) = \mathsf{G}^{\varphi_w}(\bar{S}_\ell)$, and $h+1$ is shut out in rounds $k+1,\ldots,\ell$ of φ_w. Let φ'_w be a failure pattern that is obtained from φ_w by dropping the edge $\langle (j,k),(h+1,k+1)\rangle$. We first claim that $\varphi'_w \in \Phi$. Denote by $\hat{\varphi}$ pattern that agrees with φ on $\bar{T}_\ell$, in which w is shut out in round $k+1$, and $h+1$ is shut out in rounds $k+2$ through ℓ. The lemma's statement ensures that $\hat{\varphi} \in \Phi$ for every pattern $\hat{\varphi} \in \Phi$ that agrees with φ on $\bar{T}_\ell$, in which (j,k) is shut out (in round $k+1$) and exactly one node is shut out in rounds $k+2$ through ℓ. Since $\varphi'_w \sqsubseteq \hat{\varphi}$, we have by monotonicity of Φ that $\varphi'_w \in \Phi$, as claimed.

For every process $i \neq h+1$ we have that $\mathsf{G}^{\varphi_w}(i,\ell) = \mathsf{G}^{\varphi'_w}(i,\ell)$, since G^{φ_w} and $\mathsf{G}^{\varphi'_w}$ differ only on the incoming edges of $w = (h+1,k+1)$. Since $h+1$ is shut out from time $k+1$ to ℓ, the node w appears neither in $\mathsf{G}^{\varphi_w}(i,\ell)$ nor in $\mathsf{G}^{\varphi'_w}(i,\ell)$. It follows that $(\varphi_w,\ell) \approx (\varphi'_w,\ell)$. By transitivity of $\approx$ we have that $(\varphi,\ell) \approx (\varphi'_w,\ell)$, which by Lemma 1 implies that S is covered by φ'_w. The pattern that agrees with φ'_w on S in which the nodes of S are correct is φ_{h+1}. The fact that S is covered by φ'_w at ℓ implies that $(\varphi'_w,\ell) \approx$

(φ_{h+1},ℓ), and again by transitivity of $\approx$ we have that $(\varphi,\ell) \approx (\varphi_{h+1},\ell)$, completing the inductive proof. We thus obtain that $(\varphi,\ell) \approx (\varphi_n,\ell)$. Moreover, since S is covered at ℓ by φ_n, the pattern φ_u that is guaranteed with respect to φ_n by clause (a) of the definition of hidden for $u = (j,k+1)$ satisfies $\mathsf{G}^{\varphi_u}(\bar{S}_\ell) = \mathsf{G}^{\varphi}(\bar{S}_\ell)$ and has j shut out in rounds $k+1,\ldots,n$. It follows that φ_u satisfies all three conditions (a), (b) and (c) necessary to establish the first part of the definition of $T = S \cup \{(j,k)\}$ being covered by φ at time ℓ, as required.

The second part of the proof that T is covered by φ at ℓ follows the same steps. In this case, however, we define patterns $\hat{\varphi}_m(j,k) = \varphi(j,k) \setminus \{1,\ldots,m\}$ rather than blocking edges from (j,k) in round $k+1$, we incrementally add such edges. A completely analogous inductive proof produces a pattern ψ' such that $(\varphi,\ell) \approx (\psi',\ell)$, $\mathsf{G}^{\varphi}(\bar{T}_\ell) = \mathsf{G}^{\psi'}(\bar{T}_\ell)$, and $\varphi'(j,k) = \emptyset$. By Lemma 1 we obtain that ψ' covers S at ℓ. The pattern φ' guaranteed with respect to ψ' and S satisfies that $(\varphi',\ell) \approx (\psi',\ell) \approx (\varphi,\ell)$, that $\mathsf{G}^{\varphi'}(\bar{T}_\ell) = \mathsf{G}^{\psi'}(\bar{T}_\ell) = \mathsf{G}^{\varphi}(\bar{T}_\ell)$, and that $\varphi'(j,k) = \psi'(j,k) = \emptyset$ and $\varphi'(v) = \emptyset$ for every $v \in S$. We thus have that φ' satisfies conditions (d), (e) and (f) completing the proof that $T = S \cup \{(j,k)\}$ is covered by φ at time ℓ. ■

B Upper Bound Proof

Proof of Lemma 6: In order to prove that mt-CC is a CC protocol, we have to show that Accuracy, Consistency and Completeness hold. We note that UCC satisfies these properties, as shown in [11].

Initially, $\hat{M}_x[0] = \emptyset$ and is thus vacuously accurate. By line 4 of mt-CC, the core $\hat{M}_x[k+1]$ is constructed by extending $\hat{M}_x[k]$ with events from $M_x[k]_s$, computed by UCC_s. Since UCC_s satisfies Accuracy, the resulting core $\hat{M}_x[k+1]$ is accurate, as required.

For Consistency, let r be the run of mt-CC(m,t) with adversary (φ,ζ), let $k \geq 1$, and let $\hat{M}_x[k]$ denote the core computed by mt-CC at time k in r. We proceed by induction on k. For $k = 0$, by line 0 of mt-CC we have $\hat{M}_x[k] = \hat{M}_z[k] = \emptyset$. Now assume the inductive hypothesis holds for $k-1$, i.e., that $\hat{M}_x[k-1] = \hat{M}_z[k-1]$. By line 4 of mt-CC we have $\hat{M}_x[k] = \hat{M}_x[k-1] \cup M_x[k]_s$. By Lemma 5, we have that $M_x[k]_s = M_z[k]_s$. Since $\hat{M}_x[k-1] = \hat{M}_z[k-1]$ by the inductive hypothesis, we have that $\hat{M}_x[k-1] \cup M_x[k]_s = \hat{M}_z[k-1] \cup M_z[k]_s$, and thus $\hat{M}_x[k] = \hat{M}_z[k]$. Thus, Consistency holds for mt-CC.

Finally, for Completeness, recall that $\text{UCC}(t)$ guarantees that $e \in M[s+t+1]$ for every event $e \in \mathcal{E}$ that is known to a nonfaulty process j at time s. By Lemma 5, if $m > t$ then $M_x[s+m]_s = M'_x[s+m]$ where M'_x is obtained by UCC in a run r' with adversary (φ'_s,ζ) as defined in that lemma. Since all messages delivered in r are delivered in r', it follows that if j knows e at time s in r it knows e at j in r' as well. Since $e \in M'_x[s+t+1]$ and $M'_x[s+t+1] \subseteq M'_x[s+m]$, it follows by Lemma 5 that $e \in M_x[s+m]_s$. By line 4 of mt-CC we have that $e \in \hat{M}_x[s+m]$, as desired. ■